Do Security Patterns Really Help Designers?

Koen Yskout
iMinds-DistriNet, KU Leuven
3001 Leuven, Belgium
koen.yskout@cs.kuleuven.be

Riccardo Scandariato
iMinds-DistriNet, KU Leuven
3001 Leuven, Belgium
riccardo.scandariato@cs.kuleuven.be

Wouter Joosen
iMinds-DistriNet, KU Leuven
3001 Leuven, Belgium
wouter.joosen@cs.kuleuven.be

***Abstract*—Security patterns are well-known solutions to security-specific problems. They are often claimed to benefit designers without much security expertise. We have performed an empirical study to investigate whether the usage of security patterns by such an audience leads to a more secure design, or to an increased productivity of the designers. Our study involved 32 teams of master students enrolled in a course on software architecture, working on the design of a realistically-sized banking system. Irrespective of whether the teams were using security patterns, we have not been able to detect a difference between the two treatment groups. However, the teams prefer to work with the support of security patterns.**

I. Introduction

Security patterns have been a popular research topic for the past 15 years. Like all patterns, they are expected to be general, time-tested solutions, which are distilled from real world examples and solving recurring problems. In particular, this work focusses on design-level security patterns, which have a granularity similar to, for instance, design patterns and architectural styles. Using security patterns for the design of secure software can provide value in several ways. Most importantly, the use of security patterns should result in a 'more secure' design, i.e., software systems designed with the support of security patterns should expose fewer security flaws at the design level. The objective of this work is to test this hypothesis by means of an empirical study. As an additional benefit, irrespective of possible security improvements, it seems plausible that using security patterns could increase the productivity of designers, as less time would be spent in brainstorming ad-hoc security solutions. This work analyzes this possibility as well.

Contribution. In summary, this paper presents a study answering the following two research questions:

RQ1 What is the effect of using security patterns on the security of a software design?

RQ2 What is the effect of using security patterns on the productivity of the designer?

To answer these questions, we have performed a controlled experiment with 32 teams of master students. The teams have performed 6 design tasks on a realistically-sized system in the context of a course on software architecture. As far as the target audience of security patterns is concerned, security patterns are often claimed to provide particular value to software developers with limited security expertise. For example, a popular book on security patterns states that "security patterns can be used when the people responsible for enterprises or systems have little or no security expertise" [1]. Similarly, a more recent book also claims that "the most common use for security patterns is to help application developers who are not security experts to add security in their designs" [2]. The participants of our study are not security experts, although some of them have a more security-oriented background due to their studies. Hence, they provide a very suitable sample for validating the above claims.

Approach. We have asked the teams to implement some security requirements in the design of a banking system. We have compared the resulting designs for differences between design tasks executed with and without the support of security patterns. From a methodological perspective, answering the first research question requires that we can assess the security of a software design in a quantitative way, in order to enable statistical comparison. To define such a measure in practice, we start from the consideration that security-relevant design flaws lead to security threats, which potential attackers might pose in order to exploit the flawed system. Therefore, the number of threats afflicting a given design can be seen as an effective measure of its (in)security, provided that all threats have a comparable risk value. For this reason, we have compiled a set of threats for each design task. These threats expose design-level weaknesses and, hence, a 'more secure' design should be able to counter more (if not all) of them.

Results. To our surprise, irrespective of whether the teams were using security patterns, we have not been able to detect a difference across the two treatment groups. Actually, we have observed that both groups created similar solutions.

The rest of the paper is organized as follows. In the next section, we discuss other work that is related to ours. Section III provides more details on the design of our study itself, including the precise hypotheses and measurements, followed by a description of the execution of the study in Section IV. The obtained data and our interpretation are presented in Section V. Section VI elaborates on the design approaches taken by the teams and the perception of the participants with respect to security patterns. Section VII lists the threats to validity, and our conclusion follows in Section VIII.

II. Related work

Security patterns have been a popular topic for more than fifteen years. In this period, many patterns have been published

and subsequently surveyed [3], [4], and multiple books about them have appeared [1], [2], [5].

Despite this popularity, empirical studies on security patterns are lacking. We have started to perform such studies by empirically evaluating the effect of organizing a catalog of security patterns [6]. We have found that, counter-intuitively, a richer organization does not reduce the time that is needed by a software designer to find a suitable pattern in the catalog.

In this paper, we continue along this track, albeit now with the goal of empirically confirming or refuting some alleged benefits of the security patterns themselves. While, to the best of our knowledge, no such studies have been performed before for security patterns, a sizeable body of research exists that considers the effect of well-known design patterns (like those in the 'Gang of Four' book [7]) on software design. We highlight a couple of these studies in the next paragraphs.

Most related to this paper are approaches that investigate the effect of design patterns on the number of defects in the software. An example hereof is the study performed by Vokac [8], who looked into the code of a large industrial product in order to verify whether using some well-known design patterns (e.g., observer, singleton, or factory) leads to fewer design defects. The author concludes that the usage of design patterns by itself does not necessarily lead to fewer defects, and the success depends on the context in which the patterns are used.

Besides the number of defects, patterns may also influence other dimensions of software development, such as comprehensibility and maintainability. One example of research along this line comes from Jeanmart et al. [9], who have assessed the impact of the visitor pattern on program comprehension and maintenance by means of eye tracking. They found that, while the visitor pattern does not reduce the comprehension effort, it helps when modification tasks need to be carried out.

To bundle the existing knowledge, Zhang and Budgen [10] have performed a systematic literature review that summarizes the available studies concerning the effectiveness of software design patterns. From 11 studies, they conclude that the results are not univocal, and there is in general little to no support for the claims about design patterns. From a software designer perspective, the same authors also surveyed experienced design pattern users [11], and observed that very few of the well-known design patterns were actually regarded as valuable by these users.

We conclude with the remark that security patterns are not the only technique to design secure software. Other approaches have been published (for example, UMLsec [12] and SecureUML [13]) and surveyed [14], [15].

III. Study design

All lab material related to this study, including the description of the application and the tasks, the tool that was used, as well as the (anonymized) data obtained from the participants, is available online [16].

A. Participants

The participants of the study are 64 students of a master course on software architecture taught by the authors at KU Leuven. The students are divided into 32 teams of 2 members each[1]. The participants are free to choose a teammate to work with, which is standard practice for group projects at our department.

TABLE I
Questionnaire about the training on security aspects. Possible answers were: strongly disagree (SD), disagree (D), agree (A), strongly agree (SA).

The training was sufficient to become familiar with...	SD	D	A	SA
security requirements	0	13	18	1
attacks	0	9	21	2
secure design	2	16	14	0

The participants follow different specialization trajectories within their master program, like software security, software engineering, or artificial intelligence. Most teams (26) are composed of students with different specializations. There are 7 teams where at least one member follows the security specialization.

With an anonymous survey conducted before the start of the study, we have collected additional background information from the individual participants. In this survey, about 60% of the participants consider themselves as skilled programmers with adequate background in software engineering. As UML is used as the primary language for all models in this study, we asked the participants to rate their familiarity with this modeling language: 63% of the participants rated their familiarity as either adequate or expert, 32% as limited, and 5% as beginner.

Concerning their security expertise, about 20% of the participants have rated their knowledge of security as adequate, 32% as limited, and 48% as beginner. As shown in Table I, this is corroborated by their responses to three questions regarding the security-oriented training they have received at the start of the study (see Section IV-A). Roughly one half of the teams found the training to be sufficient. Hence, the study mainly involves participants with a limited security background, which is the target audience we are interested in.

Finally, almost 60% of the participants have indicated that they have a working experience in software development. The majority of the participants has worked in a commercial software development context, i.e., the participants' background is not purely academic.

Informed consent. All students are aware of the experiment from the beginning, albeit not of the hypotheses that are tested. Throughout the entire study, the participants have the opportunity to opt out of the study. In this case, all additional information gathered by the experimenters is destroyed. Opting out does not influence the amount of work that needs to be done. All students have to carry out the tasks defined in the study, as they have a pedagogical value and are meant to

[1]These numbers do not include participants that opted out of the study (1 team), or that were removed because they did not complete the assignment according to the scheduled deadlines (4 teams).

give the students a hands-on experience of the secure design challenges.

Incentives. There are no special incentives for the participation in the study. The students receive a grade based on a written report, which summarizes their solutions to the design tasks. The grade contributes to 20% of their final mark in the course, irrespective of having opted out.

B. Application

The study is performed with the architecture of a fictitious bank as object. Throughout the entire course and prior to entering this study, the students had to develop a project using the banking system. Therefore, the banking domain is well-known to the participants by the time this study starts. Most importantly, the banking application offers sufficient architectural complexity to be non-trivial. Functionality-wise, the architecture of the banking application supports a customer to perform monetary transactions via a bank clerk, the web, or a mobile app. Also, the banking system needs to interact with other banks for executing wire transfers, and allow for real-time debit card payment processing.

The architecture is modeled using UML and documented in a 32-page PDF file. The documentation includes a context diagram, a main component-and-connector diagram, decompositions of the most important components, a deployment diagram, and documentation of the interfaces. The complete document can be found online [16].

C. Catalog of security patterns

For the study, we rely on a catalog of 36 security patterns. The catalog has been successfully used in the course for several years already and, in the past, it proved to contain a sufficiently large set of solutions for the participants to tackle their design needs. The description of each pattern in the catalog includes a brief summary, possible aliases, a problem description, a list of forces, an example, the solution (both structural and behavioral), implementation guidelines, a list of consequences and pitfalls, and known uses. The catalog can be found online [16].

D. Tasks

In order to reduce variability and to make results comparable, the participants are provided with an initial design of the banking system, as described earlier in section III-B. The design does not contain any specific security measure. The participants work in teams and have to extend and harden the design by means of seven tasks, which are labeled A through G. A summary of the tasks is given in Table II. The complete task descriptions (as provided to the participants) are available on the companion website [16].

The most important part of the description of each task depicts a security requirement that the participants have to implement in the design of the banking system. The task also includes context information, which provides the necessary background and rationale for the security requirement. Furthermore, additional constraints are included where necessary, to guide the participants and scope their effort. For instance, the context of task A is that the bank policy requires passwords to be sufficiently complex. It could be cumbersome for the employees to remember the different, complex passwords they need to access the various bank services. Therefore, the security requirement is about supporting single sign on for the bank employees. However, an additional architectural constraint is provided that interfaces towards third party services (e.g., an external fraud detection company) cannot be modified.

TABLE II
SHORT SUMMARY OF THE TASKS. TASKS ARE ASSOCIATED WITH THE SECURITY PATTERNS THAT ARE EXPECTED TO BE USEFUL.

	Task description
(A)	*(warm-up)* Reduce the number of passwords that an employee has to remember (single sign on).
B	Protect sensitive information stored by the banking app on a mobile device. *Expected pattern: Encrypted storage*
C	Prevent read access to the transaction store from a publicly available system. *Expected patterns: Demilitarized zone, Firewall*
D	Make sure customers do not have to re-authenticate for every request. *Expected patterns: Session, Session Timeout, Session Failover*
E	An employee should be accountable for all actions performed on a customer account. *Expected patterns: Secure Logger, Audit Interceptor*
F	Ensure customers can only perform operations on their own accounts. *Expected patterns: Input Guard, Authorization Enforcer*
G	Protect the data that is transmitted between a mobile device and the bank. *Expected pattern: Secure Pipe*

TABLE III
THE NUMBER OF TEAMS PER TREATMENT IS BALANCED ACROSS TASKS.

	B	C	D	E	F	G
Without patterns	14	16	17	16	18	15
With patterns	18	16	15	16	14	17

Task A is a warmup task, which is always executed as first and without using the security patterns. This task is used to familiarize the participants with the various aspects of the study, like the lab material, tools and procedures. No data is collected during this task. The other six tasks (B–G) are executed by the teams in a different, random order. The teams execute the first three tasks (of their random sequence of six tasks) using their own knowledge, and the last three tasks with the support of the catalog of security patterns. That is, each team is exposed to both treatments. The randomization has been performed with the objective of achieving a balanced amount of teams that execute each task with and without patterns. The final distribution of teams is given in Table III.

The six tasks were defined in such a way that at least one pattern from the catalog could be used to solve the task, as described in Table II. The task are of very similar difficulty and are expected to take about 45 minutes each to be completed. At the end of the study, we asked the participants to rate their perceived difficulty of tasks B–G. On average, the participants rated the six tasks with the same difficulty level (i.e., as 'average', on a five-levels scale from 'very easy' to 'very difficult').

Misuse case F.1

Short description A customer, when entering a money transfer via the web banking interface, provides a valid source account ID of which he is not a holder, thus allowing him to transfer money from some other customer's account.
Preconditions The customer is authenticated.
Attacker profile Rogue customer (via web interface)
Scenario

1) The customer enters a new transfer from one of his accounts in the web banking application.
2) Before sending the transfer to the web server, the customer changes the source account to a different, valid and existing account (e.g. by tampering the parameters of the HTTP PUT request).
3) The transaction is accepted and issued for processing.

Fig. 1. Example of a misuse case for task F (summarized).

E. Misuse cases (MUC)

Every task (except the warm-up task) is associated with 5 threats describing design-level weaknesses, which are used to test the quality of the design solutions proposed by the participants for the corresponding task. These threats have been defined before the execution of the study started. For documentation purposes, the threats are represented as misuse cases (MUC, [17]). An example of a misuse case is given in Fig. 1. As shown, each misuse case comprises a short description, a set of preconditions for the misuse case to be possible, the attacker profile, the scenario, and optionally one or more alternative scenarios. The misuse cases are chosen in such a way that they cover a broad range of weaknesses, and the overlap among them (per task) is minimal. Also, the granularity of weaknesses exposed by the misuse case is kept the same. Finally, the misuse cases associated to a task have a similar importance in terms of security risk.

F. Collected measurements

Measuring security. The most important measure that we collect is the *number of covered misuse cases* per task, which ranges from 0 to 5.

After the teams have completed all tasks and have submitted their solutions, they are provided with the misuse cases, which the experimenters use to evaluate the designs. The teams have to prepare a written report in which, per task, they discuss how their submitted solution eliminates (some of) the misuse cases or, alternatively, have to mark (some of) the misuse cases as uncovered. It is made clear to the participants that, because they are not security experts, their grade is not based on the number of covered misuse cases, but rather on the quality of their reasoning (i.e., why the misuse case is not covered, or how it is prevented by their solution).

Three security experts (one not involved with the execution of this study) independently assess the solutions. Each expert evaluates the tasks performed by about one third of the teams. The expert makes his own assessments, but takes into account the provided reasoning of the team, for example to discover undocumented but plausible assumptions made by the team. In rare cases, a discussion during the exam is used to improve the expert's understanding of the solution.

A disadvantage of this approach is that only five specific scenarios per task are considered. This is akin to unit-testing for code: even with all test cases passed, flaws may still exist. As a complementary device, and before analyzing the misuse cases, the experts make an overall assessment of the correctness of each task's solution. The *correctness score* is an ordinal variable, with 3 possible values: wrong (0), some errors (1), and correct (2). While this scoring system cannot capture fine details, and is clearly more subjective than counting the misuse cases, it is useful to obtain an additional rough estimate of the soundness of the solution. We expect that the correctness score and the number of covered misuse cases correlate.

Measuring productivity. We also collect the *time* that each team spends executing each task. This time is seamlessly and reliably measured by the tool that the teams use (see Section IV-D). The measure starts when a task is made available to the team via the tool, and ends when the solution for that task is submitted (again, via the tool). Hence, the time includes the time needed to read and analyze the task description, choose a solution, and integrate the solution into the design that is given as a starting point.

Finally, if the task is executed using security patterns, we collect the *patterns* that were selected to solve the task. This information is also obtained from the tool and is used in the discussion section of this paper.

G. Hypotheses

Given the above measurement procedure, we can translate the two research questions stated in the introduction into the following precise null hypotheses.

Security. We count how many misuse cases per task are covered by a team's solution, resulting in the security score s that can take values from 0 to 5. This leads to the following null-hypothesis that we want to disprove:
H_0^s: The usage of security patterns has no influence on the mean number of covered misuse cases for a task.

Productivity. We measure the time t it takes each of the teams to complete a security design task. This leads to the following null-hypothesis:
H_0^t: The usage of security patterns has no influence on the mean time needed to complete a task.

IV. Study execution

As mentioned earlier, our study is executed in the context of a course on software architecture. This course consists of 3 parts. In the first part, the students perform a domain analysis and elicit use cases for the banking system. In the second part, which is the main part of the course, they design the software architecture. Finally, in the third part, they revisit the design in light of security requirements. This study is positioned in the third part. As shown in Fig. 2, the study begins with a training phase and proceeds with two (non-overlapping) phases.

A. Phase 0: Training

In the training phase, the teams attend two lectures of 2.5 hours each. The lectures introduce the participants to security

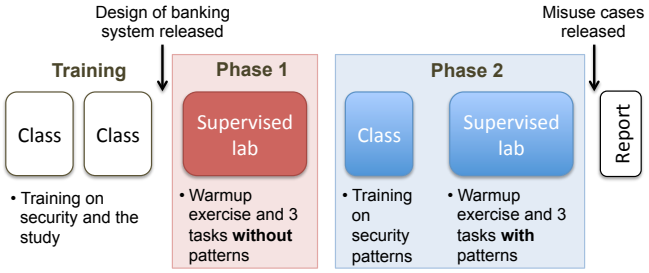


Fig. 2. The study.

and provide information about the study. For instance, the lectures cover generic security objectives like confidentiality, integrity, availability and accountability (CIAA) and illustrate common security attacks and weaknesses. Further, the lectures describe the design approach to more secure software, including secure design principles, design building blocks (e.g., encryption) and security strategies (e.g., preventing, reacting, and monitoring). The slides used during the training phase are available online [16]

Additionally, the lectures roughly explain the setup of the current study, without disclosing the tested hypotheses or mentioning the concept of security patterns, which are introduced in phase 2. Finally, the initial design of the banking system (used for the tasks) is also presented, and a demonstration is given of the tool the participants will work with. The initial design is made available in PDF to the teams right after the training is complete, and prior to the beginning of phase 1.

During the training, it is repeatedly pledged that the obtained measurements are not looked at before the final grades for the course are filed, in order to avoid influencing the participants' way of working during the execution of the study.

At the end of the lectures, the students are invited to fill out an anonymous questionnaire on their background. The answers to this questionnaire have been discussed earlier in this paper (see Section III-A).

B. Phase 1: Without security patterns

In the first phase of the study, all teams work without having access to (or knowledge of) the security patterns. The participants use their own knowledge and the information provided during the training. The teams work in a supervised lab session, where they have access to a PC with the necessary material. The teams begin with a warmup exercise in order to get familiar with the procedure, the design of the banking system, and the tool used during the study (as described in Section IV-D). The exercise consists of performing the warmup task A. Then, the teams evaluate their solution with respect to 3 misuse cases, so that they get a better understanding of how their results are evaluated (see also Section III-F).

After the warmup exercise, the teams execute 3 tasks, which are randomly selected from the 6 available tasks (B–G). The description of a task is automatically provided by the tool once the previous task has been completed and submitted. The tool also provides the description of the initial design that has to

TABLE IV
QUESTIONNAIRE AT THE END OF PHASE 1 (SD=STRONGLY DISAGREE; D=DISAGREE; A=AGREE; SA=STRONGLY AGREE).

You had a clear *understanding* of what the assignment asked you to do			
SD: 4	D: 10	A: 18	SA: 0
The training and the warmup exercise were sufficient to become familiar with the initial *architecture*			
SD: 1	D: 6	A: 22	SA: 3
TThe training and the warmup exercise were sufficient to become familiar with the *tool*			
SD: 2	D: 2	A: 21	SA: 7

be extended according to the task description. For each of the tasks, the teams restart from the initial design, i.e., whenever a task is completed, the design is reverted to the starting point before the next task is started, and their changes are not carried over to the next task. This way of working, although slightly unnatural, makes the tasks independent from one another. To increase this independence even more, the teams also have to submit their solution for each task before advancing to the next, and they are not allowed to revisit earlier tasks. The teams are also advised not to discuss the tasks with each other. Note that the misuse cases for the performed tasks are not disclosed at this stage.

At the end of phase 1, the teams fill out a questionnaire. As shown in Table IV, the answers support our assumption that the majority of the teams had a clear understanding of the assignment, and felt sufficiently familiar with the initial architectural design and the tool.

C. Phase 2: With security patterns

As shown in Fig. 2, at the start of the second phase, the teams receive a lecture on security patterns (including a brief summary of the patterns in the catalog) as well as a demonstration of the additional functionality of the tool (i.e., how to browse and select security patterns).

At the beginning of the supervised lab session, the teams perform a warmup exercise consisting in answering a question (i.e., "Which patterns from the catalog are appropriate for authentication, and why?"). In order to answer, they need to familiarize themselves with the pattern catalog and the new functionality of the tool. Afterwards, the teams perform the remaining three tasks (of their random sequence) and are requested to use security patterns to solve the tasks. Like in the previous phase, the solution for a task needs to be submitted before advancing to the next task, and going back is not allowed.

Once more, a questionnaire is administered at the end of this phase, and the answers in Table V support our assumption that the assignment was clear, and that the participants feel sufficiently familiar with the patterns.

D. Tool support

As mentioned, the execution of the study is heavily supported by a tool we built for the purpose [16]. This tool is

TABLE V
QUESTIONNAIRE AT THE END OF PHASE 2 (SD=STRONGLY DISAGREE; D=DISAGREE; A=AGREE; SA=STRONGLY AGREE).

You had a clear *understanding* of what the assignment asked you to do			
SD: 0	D: 3	A: 24	SA: 5

The training and the warmup exercise were sufficient to become familiar with the *concept of security patterns*			
SD: 0	D: 5	A: 26	SA: 1

The training and the warmup exercise were sufficient to become familiar with the *catalog* of security patterns			
SD: 1	D: 3	A: 28	SA: 0

an Eclipse environment (version 4.2.2) including the Papyrus UML editor (version 0.10M6). A custom extension provides a wizard to guide the team through the study. Also, a browser for the security pattern catalog is embedded into the tool.

When a team starts the tool for the first time, they have to enter their team code. Based on this code, the task order for that team is loaded from a configuration file. The tool enforces the sequential execution of the phases of the study, and unlocks the security pattern catalog at the start of phase 2. Furthermore, within each phase, the tool enforces the strict order of the tasks, and makes it impossible to return to an earlier task. At the start of each task, it takes care of preparing a new copy of the original architectural design.

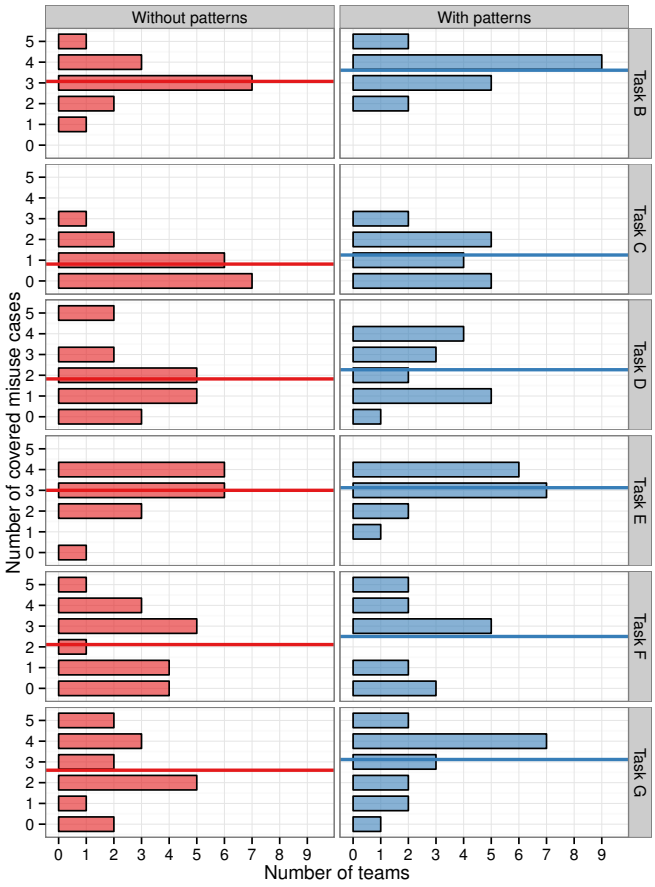
For each task, the tool measures how much time a team spends solving it. Note that the tool interrupts the time measurement when the tool loses focus or is closed, i.e., it omits periods of inactivity. For the tasks supported by security patterns, the tool additionally records which patterns are browsed (and for how long), and which are eventually selected. The tool takes care of submitting the solutions and the collected measurements to an online server at the end of each task. Throughout the study, the various questionnaires are also filled out in the tool, and submitted together with the measurements.

Finally, all configuration files and resources (e.g., the task descriptions and the pattern catalog) related to the study are encrypted to prevent participants from viewing this material outside of the tool, or at a wrong point in time. The collected measurements are encrypted as well.

V. RESULTS

In this section, the collected data is analyzed to statistically test the two hypotheses stated in Section III-G. Additional findings that are interesting but not directly related to either hypothesis can be found in Section VI. As a reminder, task A only served as a warm-up task, so all data related to this task is ignored in the rest of this paper.

For the entire study, we set the significance level $\alpha = 0.05$. When reporting p-values, we use $*$ to indicate $p < 0.05$, and $**$ for $p < 0.01$. All statistical tests are performed using R [18].



		B	C	D	E	F	G
Without patterns	μ	3.07	0.81	1.82	3.00	2.11	2.60
	σ	1.00	0.91	1.51	1.10	1.64	1.59
With patterns	μ	3.61	1.25	2.27	3.12	2.50	3.12
	σ	0.85	1.06	1.39	0.89	1.79	1.45
p-value		0.103	0.235	0.353	0.856	0.559	0.343

Fig. 3. Distribution of the number of covered misuse cases per task. The table at the bottom reports the average and standard deviation, as well as the p-values of the Mann–Whitney–Wilcoxon test.

A. Effect on security

Figure 3 shows the number of covered misuse cases per task, together with the mean per task. The mean number of covered misuse cases is consistently higher when security patterns are used, albeit not by much. We perform a two-sided Mann–Whitney–Wilcoxon (`wilcox.test` in R) test to determine whether there is indeed enough evidence to reject the null hypothesis H_0^s. The resulting p-values of the test are reported in the last row of Fig. 3. For none of the tasks can a statistically significant effect be appreciated, and hence, we cannot reject the null hypothesis.

As shown in Fig. 4, we have also compared the coverage of the individual misuse cases across the teams with and without security patterns. A two-sided test for equal proportions (`prop.test` in R) also does not detect any statistically significant difference between the two groups for any of the

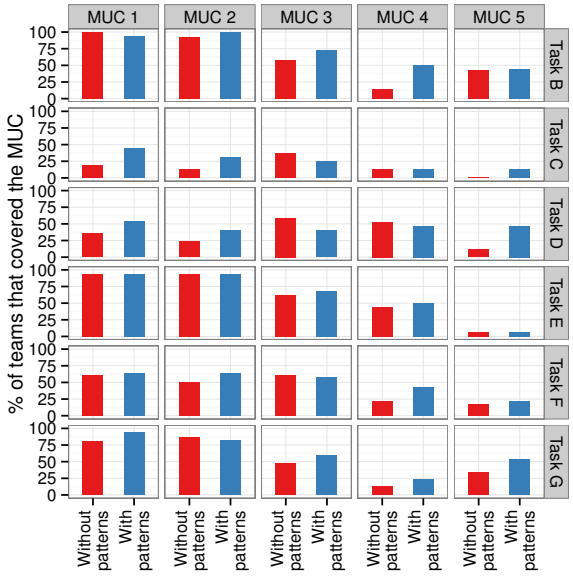
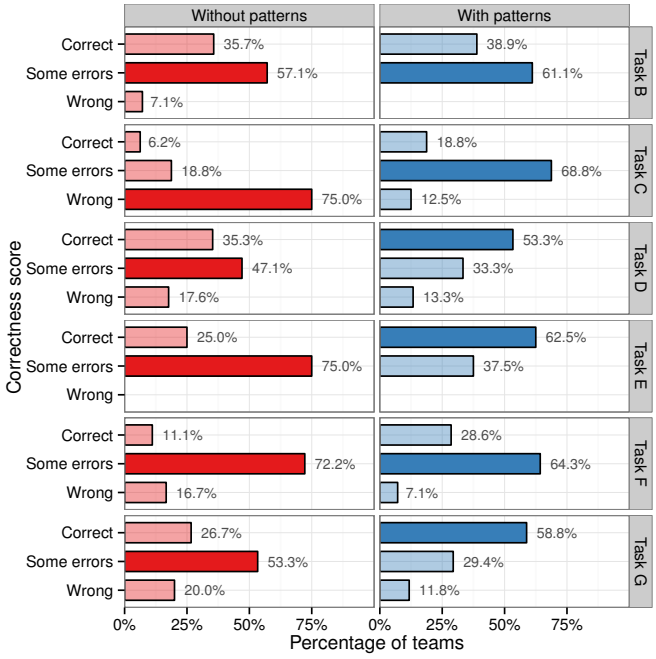


Fig. 4. Percentage of teams that have covered each individual misuse case.



	B	C	D	E	F	G
p-value	0.691	0.0012**	0.367	0.037*	0.185	0.101

Fig. 5. Distribution of the correctness scores for each task. The darker-filled bars indicate the median correctness score. The table below the graph reports the p-values of the Mann–Whitney–Wilcoxon test.

misuse cases.

Next to the number of misuse cases, we can also consider the overall correctness scores, whose distribution is given in Fig. 5. To statistically verify whether using security patterns affect the correctness score, we again perform a two-sided

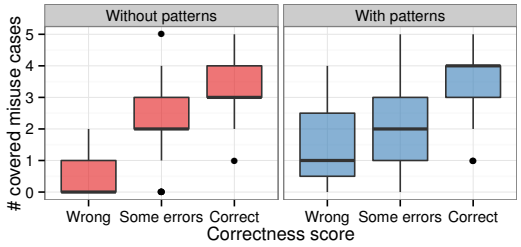


Fig. 6. Box plot aggregating all tasks and depicting the correctness score of a team's solution (x-axis) and the corresponding number of covered misuse cases (y-axis).

Mann–Whitney–Wilcoxon test. The p-values of this test (see the bottom of Fig. 5) indicate that there is a statistically significant effect for tasks C and E.

We also remark that (as expected) the correctness score and the number of covered misuse cases are positively correlated, as can be seen from Fig. 6. This is confirmed by Kendall's $\tau = 0.504$ ($p < 2 \cdot 10^{-16}$ **), where $\tau = 0$ means no association and $\tau = \pm 1$ means perfect association). This supports our assumption that both measures indeed reflect (roughly) the same phenomenon.

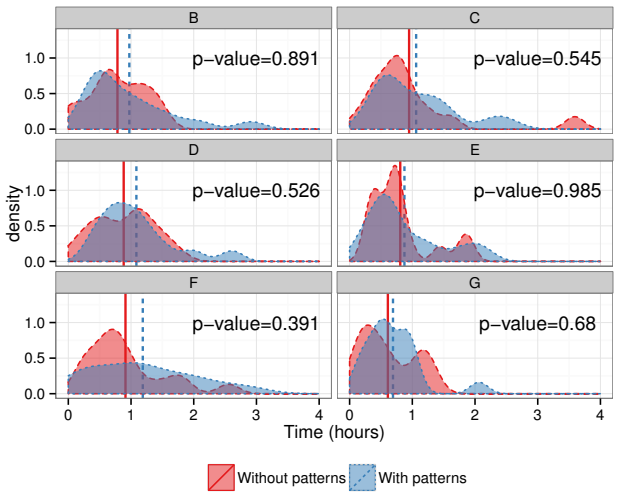
In summary, the null hypothesis H_0^s cannot be rejected, and we conclude that *the usage of security patterns cannot be shown to affect the security of the design*, when the security of the design is measured as the number of covered misuse cases. However, in a few cases, the usage of security patterns might improve the correctness of the design as a whole.

B. Effect on productivity

In some cases, individual measurements for the task execution time were suspiciously long, like a reported time of more than 4 hours for a single task (sometimes even up to multiple days). These measurements (six in total) were manually inspected and found to be erroneous due to a bug in the tool. At times, when the team left the tool running in the background without using it, the tool did not correctly register that it became inactive. Hence, these measurements were considered as outliers and consequently removed. Incidentally, we believe that this bug was only triggered in some specific instances, and did not affect the rest of the teams.

Figure 7 summarizes the time measurements for each task in a density plot. In the figure, the average execution times per task are marked by the vertical lines. In all tasks, the average time is larger when the security patterns are used (dotted blue lines). However, the difference is very small. We have also observed a quite large variance in all tasks.

To statistically assess the effect of using patterns on the execution time (and, hence, the productivity), we first test whether the data is normally distributed with the Shapiro-Wilk test. It turns out that this is not the case, so we perform a non-parametric test. In particular, a two-sample, two-sided Mann–Whitney–Wilcoxon test is used to detect the presence of a location shift (i.e., difference in median) between the time



		B	C	D	E	F	G
Without patterns	μ	0.78	0.95	0.88	0.81	0.91	0.61
	σ	0.46	0.82	0.48	0.5	0.62	0.43
With patterns	μ	0.97	1.06	1.09	0.87	1.19	0.69
	σ	0.71	0.64	0.60	0.6	0.84	0.45

Fig. 7. Density plot of the time t spent per task (B–G). The vertical lines indicate the sample means. The p-values mentioned on each plot are obtained from the Mann–Whitney–Wilcoxon test. Sample mean and standard variation are also reported in the table at the bottom of the figure.

distributions of the two groups. The test results (Fig. 7) show that no p-value lies below our significance level, so we have not observed a significant difference in the execution time. We have also performed a more general Kolmogorov–Smirnov test, which confirms the previous findings. Hence, the null hypothesis H_0^t cannot be rejected, and we conclude that *the usage of security patterns cannot be shown to affect the productivity*.

VI. Discussion

In this section, we take a deeper look into the solution strategies that were used to solve the tasks, and discuss the perception of the participants with respect to security patterns.

A. Solution strategies

An interesting question is whether the teams ended up using the same (or at least similar) solutions for each security requirement, irrespective of the treatment. To enable this analysis, we need to group similar solutions. For the cases where the teams have not used patterns, we have manually classified the solution strategies that were used into a limited number of categories. This 'taxonomy' of strategies has been developed in a bottom up, iterative way. The top rows of Figs. 8 and 9 show the percentages of each approach. It is clear that, for most tasks, there is a dominant strategy that most teams pick (this is less obvious for task B).

For the cases where the teams have used patterns, we measure the popularity of each security pattern as the percentage of teams that have chosen that pattern as (part of) their solution for a particular task. It is not uncommon that teams select a group of patterns (mostly two) to solve a requirement. The mean number of selected patterns per task are 1.6 (B), 2.2 (C), 2.1 (D), 1.8 (E), 2.6 (F), and 1.2 (G). As before, it is clear from the bottom rows of Figs. 8 and 9 that most teams align in choosing the same pattern or set of patterns. Also, the most popular patterns correspond to the patterns that were mentioned as expected in Table II.

A comparison of the most popular solutions between the treatment groups shows a clear correspondence between the two approaches. That is, when not using the patterns, the teams select solutions that resemble the ones suggested in the catalog. We remind that the treatment diffusion problem plays no role here, as the teams first use their own knowledge in phase 1 and then the patterns in a subsequent phase 2. Hence, they cannot imitate the solutions in the catalog of patterns. The correspondence of the solutions thus suggests that the solutions provided by the security patterns are already quite common among software developers, i.e., they are intuitively selected even by designers without expertise in security.

A remarkable exception occurs for task C, however. The intended solution for this task involves introducing a demilitarized zone (DMZ) using firewalls. The corresponding pattern was picked by most teams that had access to the pattern catalog, while the solutions from the teams without access to the patterns were focused more on authentication and authorization. Nevertheless, the DMZ pattern was often applied incorrectly, as can be seen from the rather low coverage of misuse cases for task C in Fig. 3. A possible explanation could be that a DMZ is not very well-known among the participants, which leads to either not thinking about using it (for the teams not using patterns), or using it incorrectly (for the teams using patterns). From Fig. 5, it can be seen that the correctness score in task C for the group with patterns is indeed centered around 'some errors', while the score for the other group is predominantly 'wrong'. This difference is statistically significant (Fig. 5).

In both treatment groups, a frequent problem with the solutions of the participants is a lack of certain details in the description of the solution. For instance, for solutions involving encryption, an aspect that was often overlooked was the key management (e.g., providing support for key setup and distribution, changing the key, or revoking a key). This was the case even when a pattern such as Encrypted Storage was used, which explicitly mentions the importance of considering key management issues [19]. The danger of omitting these details in the design is that they will then need to be specified during the actual development, possibly introducing errors or weaknesses. Possibly, the notation used in the study (UML) does not easily support the designer in expressing these concerns.

B. Perception

The questionnaire administered at end of phase 2 also included some questions to gauge the teams' perception about the usage of security patterns (Table VI).

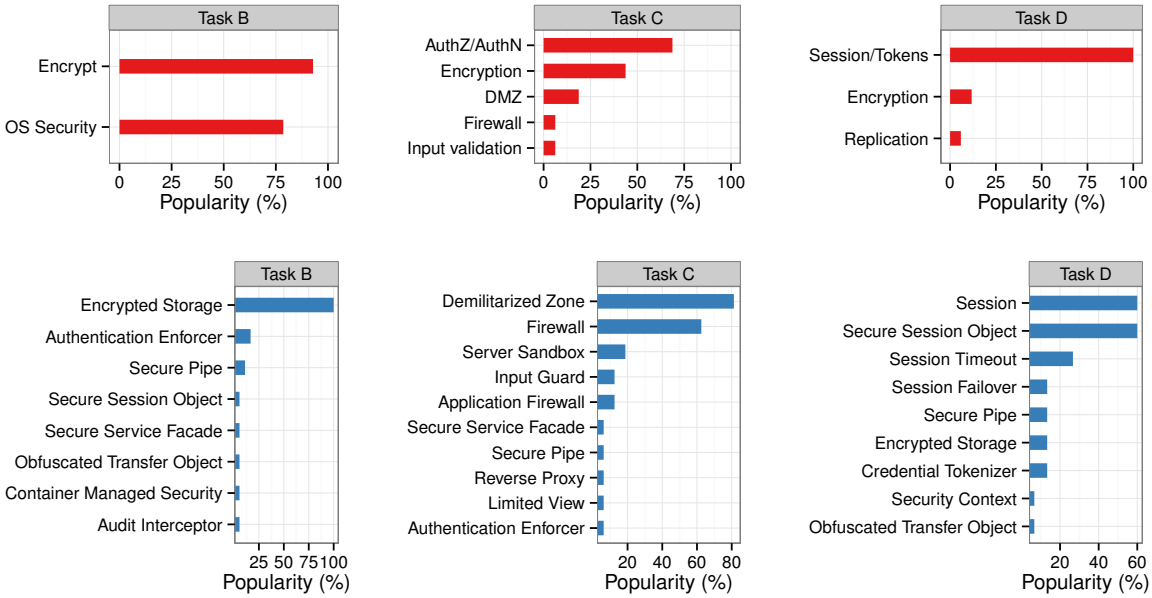


Fig. 8. Popularity of the solutions (manual solution strategies in the top row and security patterns in the bottom row), as the percentage of the teams that have implemented it, for tasks B, C, and D. Note that the teams often select more than one solution per task.

TABLE VI
PERCEPTION ABOUT SECURITY PATTERNS. ABBREVIATIONS: STRONGLY DISAGREE (SD), DISAGREE (D), AGREE (A), STRONGLY AGREE (SA).

Which technique did you like more?

No patterns:	Patterns:	No preference:
4	23	5

How hard was it to find the appropriate pattern in the catalog?

Very easy:	Easy:	Average:	Difficult:	Very difficult:
0	13	16	3	0

Are the patterns in the catalog clearly described?

SD: 0	D: 11	A: 17	SA: 4

Does the catalog contain solutions you would have not imagined yourself (i.e., are the patterns inspiring)?

SD: 1	D: 10	A: 20	SA: 1

When asked about their preference after having worked both with and without security patterns, a large majority (23 teams) stated that they prefer to work with the support of security patterns. Interestingly, all 7 teams with a member from the academic specialization in security preferred using patterns.

A possible explanation for this preference is that the presence of some guidance when solving the tasks increases the psychological confidence of the teams in their solution. The teams found the catalog easy to use: only 3 teams said the patterns were difficult to find, which is supported by our observation that most participants have indeed been able to select the intended pattern from the catalog. Moreover, according to the participants, the patterns in the catalog were clearly described. This observation agrees with our earlier study [6], which used the same pattern catalog.

Finally, a majority (21 teams) indicated that the patterns were inspiring, that is, the patterns provide solutions that the participants would not come up with themselves. This appears to contradict our observations, as we concluded that, in practice, the solutions created without access to the security patterns are often very similar to the pattern-based solutions.

VII. THREATS TO VALIDITY

Internal validity. Our study relies on randomization and the use of warmup tasks to mitigate the most important threats to internal validity. Nevertheless, some threats remain.

The misuse cases are written in a generic form, so that they apply to the solution of as many teams as possible. Nevertheless, they may not cover all vulnerabilities introduced by a solution, and thus counting the number of covered misuse cases is only an approximation for the 'true' security of the software design (insofar as this can be quantified at all).

Furthermore, Table IV shows that only a slight majority of the teams clearly understood the assignment in phase 1. This is not the case for phase 2 (Table V). The lower understanding may just be due to a lack of experience at the beginning of the study, or it may reflect a desire for more support, such as that provided by patterns.

For the time measurements, we have attempted to obtain accurate numbers by using the tool. It remains possible, however, that the tool was running in the foreground while no work was performed, leading to incorrect measurements. The measured time is thus only an upper bound on the actual time, but, from experience, our measurements are still more accurate than, e.g., relying on time sheets filled in by the participants.

External validity. There are a few threats when generalizing our results to software designers performing security-

Fig. 9. Same information as Fig. 8, but for tasks E, F, and G.

related tasks using patterns. First, the participants of our study are master students and not experienced software designers. Nevertheless, they can be considered the novices for whom security patterns are supposed to deliver the most value. Furthermore, a majority of the participants already has some professional experience. Hence, we believe that our findings can still be generalized to (beginning) professional designers.

The tasks for this study have been defined with certain security patterns in mind. While the misuse cases have not been based on these patterns, using an existing system for which the security requirements and misuse cases have already been defined would make the study more realistic. Unfortunately, the authors have been unable to find such documentation readily available in the literature.

A different threat originates from the use of an artificial tool environment. According to the participants, the used tool was not user-friendly. The main problems that were reported are bugs and performance issues of the UML editor. Also, some teams were complaining that the catalog is only available via the tool, and not on paper or online. While this is indeed an artificial situation, this was a deliberate choice in the design of the study, to make the collected data more accurate.

Further, our results reflect the case where an initial design is extended with security. Different results may be obtained for greenfield projects. However, in our experience, the approach of hardening an existing design is a rather common practice in the industrial context. Also, we have only worked with a single application, so further studies are required to confidently generalize our findings to other application domains.

Finally, our results depend on the specific selection of security patterns that are made available to the participants. The patterns in the catalog have been selected because they are well documented. Also, the catalog has been used successfully in earlier editions of the course. Hence, we are confident that the included patterns are representative of the state of the art.

VIII. Conclusion

In this study, we have set out to verify whether security patterns actually provide their touted benefits to the software designers that use them. Contrary to our expectations, we have not been able to conclusively demonstrate that the their usage improves the security of the design, nor that they influence the productivity of the software designer.

Despite these results, we remain convinced that the solutions that are proposed by the patterns are useful for software designers. It is not so clear, however, whether representing those solutions as security patterns provides sufficient added value. It appears that the strategies and solutions described by the security patterns are intuitive, also for non-expert secure software designers. Therefore, security patterns have a reduced value in pointing the designers in the right direction.

Security patterns might have an instrumental role in supporting the designers when it comes to the details of the ‘implementation’ of such strategies and solutions in a concrete design. However, this effect has not been observed either, possibly due to a sub-optimal quality of the documentation of existing security patterns, as pointed out by previous work [20]. Nevertheless, the conclusions of this study might be an incentive for the secure software engineering community to improve the state of the art in the area of security patterns.

Acknowledgments

We would like to thank Dimitri Van Landuyt, Alexander van den Berghe, and the software architecture course team for their support. This research is partially funded by the Research Fund KU Leuven, and the ADDIS project.

REFERENCES

[1] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating Security and Systems Engineering*. Wiley, December 2005.

[2] E. Fernandez-Buglioni, *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. Wiley, 2013.

[3] N. Yoshioka, H. Washizaki, and K. Maruyama, "A survey on security patterns," *Progress in Informatics*, vol. 5, no. 5, pp. 35–47, 2008.

[4] A. V. Uzunov, E. B. Fernandez, and K. Falkner, "Securing distributed systems using patterns: A survey," *Computers & Security*, vol. 31, no. 5, pp. 681–703, 2012.

[5] C. Steel and R. Nagappan, *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Pearson Education, 2006.

[6] K. Yskout, R. Scandariato, and W. Joosen, "Does organizing security patterns focus architectural choices?," in *Software Engineering (ICSE), 2012 34th International Conference on*, pp. 617–627, IEEE, 2012.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[8] M. Vokac, "Defect frequency and design patterns: an empirical study of industrial code," *Software Engineering, IEEE Transactions on*, vol. 30, pp. 904–917, Dec 2004.

[9] S. Jeanmart, Y.-G. Gueheneuc, H. Sahraoui, and N. Habra, "Impact of the visitor pattern on program comprehension and maintenance," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 69–78, IEEE Computer Society, 2009.

[10] C. Zhang and D. Budgen, "What do we know about the effectiveness of software design patterns?," *Software Engineering, IEEE Transactions on*, vol. 38, no. 5, pp. 1213–1231, 2012.

[11] C. Zhang and D. Budgen, "A survey of experienced user perceptions about software design patterns," *Information and Software Technology*, vol. 55, pp. 822–835, May 2013.

[12] J. Jürjens, *Secure systems development with UML*. Springer, 2005.

[13] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security: From uml models to access control infrastructures," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 15, no. 1, pp. 39–91, 2006.

[14] P. Nguyen, J. Klein, Y. Le Traon, and M. Kramer, "A systematic review of model-driven security," in *20th Asia-Pacific Software Engineering Conference (APSEC'13)*, vol. 1, pp. 432–441, Dec 2013.

[15] J. Jensen and M. G. Jaatun, "Security in model driven development: A survey," in *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pp. 704–709, IEEE, 2011.

[16] "Companion website." http://people.cs.kuleuven.be/~koen.yskout/icse15.

[17] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requirements engineering*, vol. 10, no. 1, pp. 34–44, 2005.

[18] "The R Project for Statistical Computing." http://www.r-project.org/.

[19] D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt, "Security patterns repository version 1.0," 2002.

[20] T. Heyman, K. Yskout, R. Scandariato, and W. Joosen, "An analysis of the security patterns landscape," in *IEEE Workshop on Software Engineering for Secure Systems (SESS)*, 2007.